```
import numpy as np

def bankers_algorithm(Max, Need, Available, Allocated, Request, process_id):
    """
    银行家算法实现

    参数:
    Max: 最大需求矩阵，形状为(n_processes, n_resources)
    Need: 还需要的资源矩阵，形状为(n_processes, n_resources)
    Available: 可用资源向量，形状为(n_resources,)
    Allocated: 已分配资源矩阵，形状为(n_processes, n_resources)
    Request: 资源请求向量，形状为(n_resources,)
    process_id: 发出请求的进程 ID

    返回:
    tuple: (是否可以分配, 更新后的 Available, 更新后的 Max, 更新后的 Need, 更新后的 Allocated)
    """
    n_processes, n_resources = Max.shape

    # 打印初始状态
    print("===== 银行家算法资源分配 =====")
    print(f"\n 进程 ID: {process_id}")
    print(f"请求资源: {Request}")

    print("\n 初始状态:")
    print("可用资源 (Available):")
    print(Available)
    print("\n 最大需求矩阵 (Max):")
    print(Max)
    print("\n 还需要的资源矩阵 (Need):")
    print(Need)
    print("\n 已分配资源矩阵 (Allocated):")
    print(Allocated)

    # 步骤 1: 检查请求是否超过 Need
    if np.any(Request > Need[process_id]):
        print("\n 错误：进程请求的资源超过了它声明的最大需求")
        print(f"进程 {process_id} 的最大需求: {Max[process_id]}")
        print(f"进程 {process_id} 已分配资源: {Allocated[process_id]}")
        print(f"进程 {process_id} 还需要资源: {Need[process_id]}")
        print(f"进程 {process_id} 请求资源: {Request}")
        return False, Available, Max, Need, Allocated
```

```
    print("\n 步骤 1 通过：请求未超过进程声明的最大需求")

    # 步骤 2: 检查请求是否超过可用资源
    if np.any(Request > Available):
        print("\n 错误：进程请求的资源超过了系统当前可用的资源")
        print(f"系统当前可用资源: {Available}")
        print(f"进程 {process_id} 请求资源: {Request}")
        return False, Available, Max, Need, Allocated

    print("\n 步骤 2 通过：请求未超过系统当前可用资源")

    # 步骤 3: 尝试分配资源
    print("\n  尝试分配资源...")
    Available_temp = Available - Request
    Allocated_temp = Allocated.copy()
    Allocated_temp[process_id] += Request
    Need_temp = Need.copy()
    Need_temp[process_id] -= Request

    print("\n 尝试分配后的临时状态:")
    print("临时可用资源 (Available_temp):")
    print(Available_temp)
    print("\n 临时已分配资源矩阵 (Allocated_temp):")
    print(Allocated_temp)
    print("\n 临时还需要的资源矩阵 (Need_temp):")
    print(Need_temp)

    # 步骤 4: 安全性检查
    print("\n  执行安全性检查...")
    safe, safe_sequence, safety_steps = is_system_safe(Available_temp, Max, Need_temp,
Allocated_temp)

    # 打印安全性检查的详细步骤
    print("\n 安全性检查详细步骤:")
    for i, step in enumerate(safety_steps):
        print(f"\n 迭代 {i+1}:")
        print(f"  找到进程 {step['process']} 可以完成")
        print(f"  当前 Work: {step['work']}")
        print(f"  进程 {step['process']} 的 Allocated: {step['allocated']}")
        print(f"  新的 Work: {step['new_work']}")

    if safe:
        print("\n 步骤 4 通过：系统处于安全状态")
        print(f"安全序列: {safe_sequence}")
```

```
            # 步骤 5: 确认分配
            print("\n    确认分配资源...")
            return True, Available_temp, Max, Need_temp, Allocated_temp
        else:
            print("\n 步骤 4 失败：系统处于不安全状态")
            print("不能分配资源，保持系统原状态")
            return False, Available, Max, Need, Allocated

def is_system_safe(Available, Max, Need, Allocated):
    """
    检查系统是否处于安全状态

    参数:
    Available: 可用资源向量
    Max: 最大需求矩阵
    Need: 还需要的资源矩阵
    Allocated: 已分配资源矩阵

    返回:
    tuple: (是否安全, 安全序列, 安全检查步骤)
    """
    n_processes, n_resources = Max.shape
    Work = Available.copy()
    Finish = np.zeros(n_processes, dtype=bool)
    safe_sequence = []
    safety_steps = []

    print(f"\n 初始状态  - Work: {Work}, Finish: {Finish}")

    # 寻找安全序列
    iteration = 1
    while True:
        print(f"\n 迭代  {iteration}:")
        found = False
        for i in range(n_processes):
            if not Finish[i] and np.all(Need[i] <= Work):
                print(f"    找到进程  {i}  满足  Need[{i}] <= Work")
                print(f"    Need[{i}]: {Need[i]}, Work: {Work}")

                # 进程 i 可以完成
                old_work = Work.copy()
                Work += Allocated[i]
                Finish[i] = True
```

```
                safe_sequence.append(i)

                # 记录安全检查步骤
                safety_steps.append({
                    'process': i,
                    'work': old_work,
                    'allocated': Allocated[i],
                    'new_work': Work
                })

                print(f"    进程 {i} 可以完成")
                print(f"    Work 更新为: {Work}")
                print(f"    Finish 更新为: {Finish}")

                found = True
                break

        if not found:
            print("\n    没有找到可以完成的进程")
            break

        iteration += 1

    # 检查是否所有进程都能完成
    if np.all(Finish):
        print("\n 安全性检查通过：所有进程都可以完成")
        return True, safe_sequence, safety_steps
    else:
        print("\n 安全性检查失败：存在无法完成的进程")
        unfinished = [i for i in range(n_processes) if not Finish[i]]
        print(f"    未完成的进程: {unfinished}")
        return False, [], safety_steps

# 测试示例
def test_bankers_algorithm():
    # 定义示例数据
    Max = np.array([
        [7, 5, 3],
        [3, 2, 2],
        [9, 0, 2],
        [2, 2, 2],
        [4, 3, 3]
    ])
```

```
Allocated = np.array([
    [0, 1, 0],
    [2, 0, 0],
    [3, 0, 2],
    [2, 1, 1],
    [0, 0, 2]
])

Need = Max - Allocated

Available = np.array([3, 3, 2])

# 测试案例 1: 进程 1 请求资源[1, 0, 2]
print("\n\n===== 测试案例 1: 进程 1 请求资源[1, 0, 2] =====")
Request = np.array([1, 0, 2])
process_id = 1

# 调用银行家算法
can_allocate, new_Available, new_Max, new_Need, new_Allocated = bankers_algorithm(
    Max, Need, Available, Allocated, Request, process_id
)

# 输出结果
print("\n\n 资源分配结果:")
print(f"是否可以分配: {can_allocate}")
if can_allocate:
    print("\n 分配后的资源状态:")
    print("可用资源 (Available):")
    print(new_Available)
    print("\n 最大需求矩阵 (Max):")
    print(new_Max)
    print("\n 还需要的资源矩阵 (Need):")
    print(new_Need)
    print("\n 已分配资源矩阵 (Allocated):")
    print(new_Allocated)

# 测试案例 2: 进程 4 请求资源[3, 3, 0]
print("\n\n===== 测试案例 2: 进程 4 请求资源[3, 3, 0] =====")
Request = np.array([3, 3, 0])
process_id = 4

# 调用银行家算法
can_allocate, new_Available, new_Max, new_Need, new_Allocated = bankers_algorithm(
    Max, Need, Available, Allocated, Request, process_id
```

```
        )

        # 输出结果
        print("\n\n 资源分配结果:")
        print(f"是否可以分配: {can_allocate}")
        if can_allocate:
            print("\n 分配后的资源状态:")
            print("可用资源 (Available):")
            print(new_Available)
            print("\n 最大需求矩阵 (Max):")
            print(new_Max)
            print("\n 还需要的资源矩阵 (Need):")
            print(new_Need)
            print("\n 已分配资源矩阵 (Allocated):")
            print(new_Allocated)

if __name__ == "__main__":
    test_bankers_algorithm()
```

代码运行结果如下：

```
运行结果===== 测试案例 1: 进程 1 请求资源[1, 0, 2] =====
===== 银行家算法资源分配 =====

进程 ID: 1
请求资源: [1 0 2]

初始状态:
可用资源 (Available):
[3 3 2]

最大需求矩阵 (Max):
[[7 5 3]
 [3 2 2]
 [9 0 2]
 [2 2 2]
 [4 3 3]]

还需要的资源矩阵 (Need):
[[7 4 3]
 [1 2 2]
 [6 0 0]
 [0 1 1]
 [4 3 1]]

已分配资源矩阵 (Allocated):
```

[[0 1 0]
 [2 0 0]
 [3 0 2]
 [2 1 1]
 [0 0 2]]

步骤 1 通过：请求未超过进程声明的最大需求

步骤 2 通过：请求未超过系统当前可用资源

尝试分配资源...

尝试分配后的临时状态:
临时可用资源 (Available_temp):
[2 3 0]

临时已分配资源矩阵 (Allocated_temp):
[[0 1 0]
 [3 0 2]
 [3 0 2]
 [2 1 1]
 [0 0 2]]

临时还需要的资源矩阵 (Need_temp):
[[7 4 3]
 [0 2 0]
 [6 0 0]
 [0 1 1]
 [4 3 1]]

执行安全性检查...

初始状态 - Work: [2 3 0], Finish: [False False False False False]

迭代 1:
  找到进程 1 满足 Need[1] <= Work
  Need[1]: [0 2 0], Work: [2 3 0]
  进程 1 可以完成
  Work 更新为: [5 3 2]
  Finish 更新为: [False True False False False]

迭代 2:
  找到进程 3 满足 Need[3] <= Work
  Need[3]: [0 1 1], Work: [5 3 2]

进程 3 可以完成
Work 更新为: [7 4 3]
Finish 更新为: [False True False True False]

迭代 3:
找到进程 0 满足 Need[0] <= Work
Need[0]: [7 4 3], Work: [7 4 3]
进程 0 可以完成
Work 更新为: [7 5 3]
Finish 更新为: [True True False True False]

迭代 4:
找到进程 2 满足 Need[2] <= Work
Need[2]: [6 0 0], Work: [7 5 3]
进程 2 可以完成
Work 更新为: [10 5 5]
Finish 更新为: [True True True True False]

迭代 5:
找到进程 4 满足 Need[4] <= Work
Need[4]: [4 3 1], Work: [10 5 5]
进程 4 可以完成
Work 更新为: [10 5 7]
Finish 更新为: [True True True True True]

迭代 6:

没有找到可以完成的进程

安全性检查通过：所有进程都可以完成

安全性检查详细步骤:

迭代 1:
找到进程 1 可以完成
当前 Work: [2 3 0]
进程 1 的 Allocated: [3 0 2]
新的 Work: [10 5 7]

迭代 2:
找到进程 3 可以完成
当前 Work: [5 3 2]
进程 3 的 Allocated: [2 1 1]
新的 Work: [10 5 7]

迭代 3:
找到进程 0 可以完成
当前 Work: [7 4 3]
进程 0 的 Allocated: [0 1 0]
新的 Work: [10 5 7]

迭代 4:
找到进程 2 可以完成
当前 Work: [7 5 3]
进程 2 的 Allocated: [3 0 2]
新的 Work: [10 5 7]

迭代 5:
找到进程 4 可以完成
当前 Work: [10 5 5]
进程 4 的 Allocated: [0 0 2]
新的 Work: [10 5 7]

步骤 4 通过：系统处于安全状态
安全序列: [1, 3, 0, 2, 4]

确认分配资源...

资源分配结果:
是否可以分配: True

分配后的资源状态:
可用资源 (Available):
[2 3 0]

最大需求矩阵 (Max):
[[7 5 3]
[3 2 2]
[9 0 2]
[2 2 2]
[4 3 3]]

还需要的资源矩阵 (Need):
[[7 4 3]
[0 2 0]
[6 0 0]
[0 1 1]

[4 3 1]]

已分配资源矩阵 (Allocated):
[[0 1 0]
[3 0 2]
[3 0 2]
[2 1 1]
[0 0 2]]

===== 测试案例 2: 进程 4 请求资源[3, 3, 0] =====
===== 银行家算法资源分配 =====

进程 ID: 4
请求资源: [3 3 0]

初始状态:
可用资源 (Available):
[3 3 2]

最大需求矩阵 (Max):
[[7 5 3]
[3 2 2]
[9 0 2]
[2 2 2]
[4 3 3]]

还需要的资源矩阵 (Need):
[[7 4 3]
[1 2 2]
[6 0 0]
[0 1 1]
[4 3 1]]

已分配资源矩阵 (Allocated):
[[0 1 0]
[2 0 0]
[3 0 2]
[2 1 1]
[0 0 2]]

步骤 1 通过：请求未超过进程声明的最大需求

步骤 2 通过：请求未超过系统当前可用资源

尝试分配资源...

尝试分配后的临时状态:
临时可用资源 (Available_temp):
[0 0 2]

临时已分配资源矩阵 (Allocated_temp):
[[0 1 0]
 [2 0 0]
 [3 0 2]
 [2 1 1]
 [3 3 2]]

临时还需要的资源矩阵 (Need_temp):
[[7 4 3]
 [1 2 2]
 [6 0 0]
 [0 1 1]
 [1 0 1]]

执行安全性检查...

初始状态 - Work: [0 0 2], Finish: [False False False False False]

迭代 1:

  没有找到可以完成的进程

安全性检查失败：存在无法完成的进程
  未完成的进程: [0, 1, 2, 3, 4]

安全性检查详细步骤:

步骤 4 失败：系统处于不安全状态
不能分配资源，保持系统原状态

资源分配结果:
是否可以分配: False